Strong Turing Completeness of Continuous Chemical Reaction Networks and Compilation of Mixed Analog-Digital Programs

François Fages, Guillaume Le Guludec, Olivier Bournez, Amaury Pouly

Strong Turing Completeness of Continuous Chemical Reaction Networks and Compilation of Mixed Analog-Digital Programs

François Fages[1], Guillaume Le Guludec[1,2],
Olivier Bournez[3], and Amaury Pouly[4]

[1] Inria, Université Paris-Saclay, EP Lifeware, Palaiseau, France
[2] Sup Telecom, Paris, France
[3] LIX, CNRS, Ecole Polytechnique, Palaiseau, France
[4] Max Planck Institute for Computer Science, Saarbrücken, Germany

Abstract. When seeking to understand how computation is carried out in the cell to maintain itself in its environment, process signals and make decisions, the continuous nature of protein interaction processes forces us to consider also analog computation models and mixed analog-digital computation programs. However, recent results in the theory of analog computability and complexity establish fundamental links with classical programming. In this paper, we derive from these results the strong (uniform computability) Turing completeness of chemical reaction networks over a finite set of molecular species under the differential semantics, solving a long standing open problem. Furthermore we derive from the proof a compiler of mathematical functions into elementary chemical reactions. We illustrate the reaction code generated by our compiler on trigonometric functions, and on various sigmoid functions which can serve as markers of presence or absence for implementing program control instructions in the cell and imperative programs. Then we start comparing our compiler-generated circuits to the natural circuit of the MAPK signaling network, which plays the role of an analog-digital converter in the cell with a Hill type sigmoid input/output functions.

1 Introduction

"The varied titles of Turing's published work disguise its unity of purpose. The central problem with which he started, and to which he constantly returned, is the extent and the limitations of mechanistic explanations of nature.", Max Newman.

The Church-Turing thesis states that there is only one notion of effective computation over discrete structures (integers, words, ...), and in fact all mechanistic computation models devised up to now (Church's λ-calculus, Post's rewriting systems, random access machines, programming languages,...) have always been shown to be encodable in Turing machines. The more recent physical Church-Turing thesis goes beyond the original thesis by stating that all physically computable functions are Turing-computable.

In this view, it is theoretically possible to give a computational meaning to information processing in the cell in terms of algorithms and programs. However, while one lesson of Computer Science is that digital computation scales up to very large circuits and programs, contrarily to analog computation, one has to face the paradox that in a cell, even if one can observe an all-or-nothing activation of genes, one cannot deny the importance of the continuous gradual activations of protein complexes, of the time it takes, of the absence of clock signals, i.e. the importance of analog computation in the cell [20,43,41].

Classical computability and complexity theories mainly focus on computation over discrete domains, i.e. words or integers. When dealing with reals or functions, several approaches can be considered. In computational analysis, the notion of computation over the real numbers is defined in terms of approximation in arbitrary but finite precision:

Definition 1 ([48]). *A real number $r \in \mathbb{R}$ is computable (resp. in polynomial time) in the sense of computational analysis if there exists an effective approximation program of r in arbitrary precision, i.e. a Turing machine which takes as input a precision $p \in \mathbb{N}$ and outputs a rational number $r_p \in \mathbb{Q}$ s.t. $|r - r_p| \leq 2^{-p}$ (resp. in a time polynomial in p).*

Clearly, every real number can be represented as an infinite string representing a converging Cauchy sequence as above, and a computable real is one whose representation is computable. In this setting, a computable real number can thus be seen as a program which takes as input an accuracy, and returns as output an approximation of the real number by a rational number at the requested precision. A computable function is then a program that maps any (computable or not[5]) approximation of a real x to an approximation of $f(x)$.

Definition 2 ([48]). *A function $f : \mathbb{R} \to \mathbb{R}$ is computable if there exists a Turing machine with oracle which computes an approximation of $f(x)$ given x as oracle. It is computable in polynomial time if this is done in a time polynomial in p and m for $x \in [-2^m, 2^m]$.*

In this paper, we consider these notions to give a mathematical meaning to the notion of biochemical computation with continuous concentrations. In this view, the language of biochemical reactions is seen as a programming language for computing with non negative real valued concentrations, i.e. over $\mathbb{R}_+$. We consider elementary reactions, i.e. reactions with at most two reactants and with mass-action-law kinetics. It is well known that the other classical biochemical rate functions, such as Michaelis-Menten, Hill kinetics, are derived by reduction of elementary reaction systems with mass-action law kinetics, using for instance quasi-steady state or quasi equilibrium approximations [44].

We first show the Turing completeness in the strong sense of uniform computability, of elementary biochemical reactions without polymerisation under

[5] Restricting the definition to computable arguments might seem quite natural but is not the classical definition of computable analysis, see the appendix of [48].

the differential semantics on a finite universe of molecular species. This solves an open problem explicitly mentionned in [17]. where it was shown that a Turing machine could be simulated by a chemical reaction network with a small probability of error. Although not surprising, this result is in sharp contrast to the discrete semantics of reaction systems which are not Turing complete without either the tolerance of a small probability error [17], or the addition of other mechanisms such as the unbounded dynamic creation of membranes [2,38,9], or the presence of polymerization reactions on an infinite universe of polymers [10] or DNA stacks [40].

Furthermore, following [6] we generalize the purely analog characterization of the complexity class PTIME to positive binary reaction systems which stabilize on one component with a trajectory length bounded by a polynomial of the input and the precision.

Then we derive from the proof of these results a compiler of behavioural specifications[6] into elementary reaction systems, without prejudging of their biochemical implementation, by enzymatic reactions [37], DNA [13] or RNA for instance.

We illustrate this approach with the compilation of trigonometric functions, such as the the cosine function, as either functions of time or of an input variable. Then, we study different sigmoid functions which can serve as markers of presence or absence for implementing program control instructions and compiling imperative programs.

Then we start comparing our compiler-generated circuits to natural circuits, with the example of the MAPK signaling network, which plays the role of an analog-digital converter in the cell with a Hill type sigmoid input/output function [28].

2 Computational Functions and Computational Complexity over the Reals

The *General Purpose Analog Computer* (GPAC) of Shannon [46] is a model of computation based on circuits built from analog blocks. A set of variables or entries $x, y, z, \ldots$ including time t are considered and four types of blocks (constants, sums, products, and Stieltjes integral of one variable with respect to another variable - by default the time variable when it is not indicated) are connected (with possibly feedback connexions) in order to generate a system whose dynamic is considered as "generating" functions. Shannon's original presentation suffers from several problems, including the fact that some circuits may or may not have a solution. This problem was solved in [26] which gives a satisfactory definition of GPAC-generable functions in terms of the solution to polynomial initial value problems in polynomial differential equations (PIVP):

[6] For the sake of reproducibility, all the examples described in this paper are directly executable online in Biocham v4 (`http://lifeware.inria.fr/biocham4`) notebooks available at `http://lifeware.inria.fr/wiki/software/#CMSB17`.

Definition 3. *[26] A function* $f : \mathbb{R}_+ \to \mathbb{R}$ *is* GPAC-generable[7] *if it is one component of the* $y(t)$ *solution of some ordinary differential equation* $y'(t) = p(y(t))$ *for a polynomial vector* $p \in \mathbb{R}^n[\mathbb{R}^n]$ *and initial values* $y(0) \in \mathbb{R}^n$.

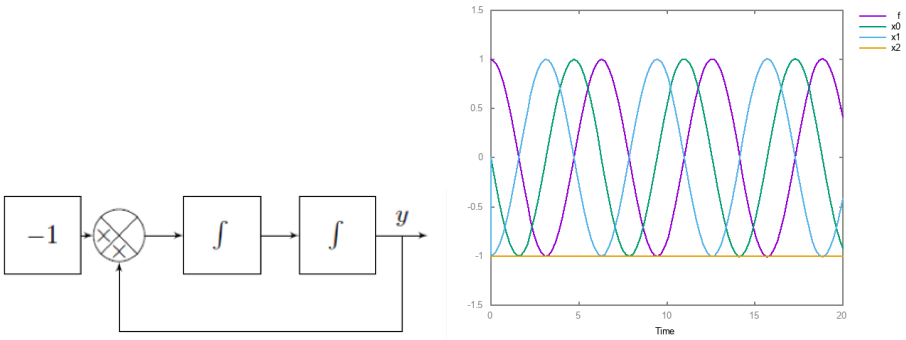

Fig. 1. GPAC circuit for generating the cosine function as a function of time, and numerical simulation trace.

For example, the GPAC (`y = integral integral -1 * y`) shown in Figure 1 is constructed with two integral blocks and a multiplication by -1 which gives $y''(t) = -y(t)$. This circuit when initialized with $y(0) = 1$ *generates* the cosine function, $cos(t)$, as a function of time. The class of GPAC-generable functions enjoys a number of properties, such as stability by addition, multiplication and composition, and also contains elementary functions such as trigonometric functions, exponential functions, logarithms, etc. This notion of generability has for some time been considered synonymous with analog-computability, which made the GPAC a computation model less expressive than computational analysis as some functions such as Rieman's Zeta function or Euler's Gamma functions are known not to be differentially algebraic [46].

However, it is possible to define a notion of GPAC-computability which is both natural in terms of PIVP and equivalent to computational analysis. The idea is to proceed by approximation of the result for any entry on one component of the system, as follows:

Definition 4. *[4] A function* $f : \mathbb{R} \to \mathbb{R}$ *is* GPAC-computable *if there are polynomial vectors* $p \in \mathbb{R}^n[\mathbb{R}^n]$, *a polynomial* $q \in \mathbb{R}^n[\mathbb{R}]$ *such that for all* x *there exists some (necessarily unique) function* $y : \mathbb{R} \to \mathbb{R}^n$ *such that*

$$y(0) = q(x),\ y'(t) = p(y(t))$$

and $|y_1(t) - f(x)| \leq y_2(t)$, *with* $y_2(t) \geq 0$ *decreasing and* $\lim_{t\to\infty} y_2(t) = 0$.

[7] This definition can be generalized to functions of several variables over different domains [7].

In other words, the computation of f with the argument x consists in putting the system in a polynomially dependent state of x, then letting the system evolve according to the dynamics described by p. The result of the computation is obtained in one component of the system, say the first, with arbitrary precision given by some other component of the system, say the second, is decreasing[8] to 0.

Then the following theorem perfectly reconciles the notions of digital (i.e. by Turing machines) and analog (i.e. by PIVP) computability:

Theorem 1. *[4,5] A function is computable in the sense of computational analysis if and only if it is GPAC-computable.*

While previous result is conciliating both notions at the computability level, such a result was missing at the complexity level. A clear difficulty is that a naive definition of the complexity in terms of the time necessary to reach a given precision can not be appropriate, since it is always possible to contract time in a PIVP by a change of the time variable, e.g. $t_{\text{fast}} = e^t$, and multiply the differential equations by an arbitrary term.

This has been solved recently in [39] by demonstrating that taking the *length of the trajectory* as measure of computational complexity, i.e. a combination of time and space (amplitude), which takes into account the cost of computing for instance $t_{\text{fast}} = e^t$, yields a valid notion of time complexity, equivalent to classical time complexity. In particular, a purely analog characterization of the complexity class PTIME has been given in [6]. Let $||y||$ refers to the infinite norm of y (i.e. the maximum absolute value of its components).

Definition 5. *[6] A function $f : \mathbb{R} \to \mathbb{R}$ is said to be Ω -computable in length, where $\Omega : \mathbb{R}_+^2 \to \mathbb{R}$, if there are polynomial vectors $p \in \mathbb{R}^n[\mathbb{R}^n]$, a polynomial $q \in \mathbb{R}^n[\mathbb{R}]$ such that for all x there exists some (necessarily unique) function $y : \mathbb{R} \to \mathbb{R}^n$ satisfying for all $t \in \mathbb{R}_+$:*

- *$y(0) = q(x)$ and $y'(t) = p(y(t))$ with $||y'(t)|| \geq 1$ (holds if t is one variable),*
- *for any μ, if $\int_0^t ||y'(\tau)|| \, \mathrm{d}\tau \geq \Omega(|x|, \mu)$ then $|y_1(t) - f(x)| \leq e^{-\mu}$.*

Theorem 2. *[6] The Ω-computable functions in length, where Ω is a polynomial, are exactly the functions computable in polynomial time in the sense of the computational analysis.*

Taking unrestricted Ω leads back to the previous notion of computable functions in the sense of computational analysis.

Theorem 1 implies in particular that polynomial differential equations (PIVP) are universal. This is in a strong sense, compared to notions of universality used in articles such as [34,27] where it is basically shown that boolean circuits can be realized, yielding a *non-uniform* notion of computability: for each input there

[8] The decreasing assumption is here to yield a simple way to decide when the result on the first component is correct with the required precision: given some precision ϵ, just wait until the second component is less than ϵ.

exists an ODE system computing the result. Here this is a *uniform computability* result: a given polynomial differential equation is able to simulate a Turing machine on all inputs, independently of the size of the input.

In order to be more concrete on the encoding of Turing machines, let us rephrase [6]. One can fix a finite alphabet $\Gamma = \{0, .., k-2\}$ and encode a word $w = w_1 w_2 \dots w_{|w|}$ by the couple $\psi(w) = \left(\sum_{i=1}^{|w|} w_i k^{-i}, |w|\right)$. There is nothing special about this encoding, other encodings may be used, however, two crucial properties are necessary: (i) $\psi(w)$ must provide a way to recover the word without ambiguity, (ii) $||\psi(w)||$ is $O(|w|)$. In particular, over the alphabet $\Gamma = \{0, 1\}$, the use of base 3 (instead of base 2) simplifies the decoding.

Now consider any decision problem (language) $\mathcal{L} \subset \Gamma^*$. If $\mathcal{L}$ is decidable, then there is a Turing machine that decides it. Then [6] provides (effectively from the Turing machine) some polynomial vectors $p \in \mathbb{R}^n[\mathbb{R}^n]$ and a polynomial $q \in \mathbb{R}^n[\mathbb{R}]$ such that for all $w \in \Gamma^*$ there is a (unique) $y : \mathbb{R}_+ \to \mathbb{R}^d$ such that for all $t \in \mathbb{R}_+$:

1. $y(0) = q(\psi(w))$ and $y'(t) = p(y(t))$ with $||y'(t)|| \geq 1$,
2. if $|y_1(t)| \geqslant 1$ for some t then $|y_1(u)| \geqslant 1$ for all $u \geqslant t$ (the decision is stable)
3. if $w \in \mathcal{L}$ (resp. $\notin \mathcal{L}$) then there is some t with $y_1(t) \geqslant 1$ (resp. $\leqslant -1$)

Furthermore, if $\mathcal{L}$ is decided in polynomial time (i.e. is in class PTIME) then there is some polynomial Ω (that can be obtained effectively from the polynomial bound for $\mathcal{L}$ and from the Turing machine) such that this happens in polynomial length: condition 3. is replaced by

3. if $w \in \mathcal{L}$ (resp. $\notin \mathcal{L}$) and $\int_0^t ||y'(\tau)|| \, \mathrm{d}\tau \geq \Omega(|w|)$ then $y_1(t) \geqslant 1$ (resp. $\leqslant -1$)

In other words, [6] is considering a notion of termination given by the fact that some variable becomes of absolute value greater than 1: if the value is greater than 1 (repectively: less than -1) this corresponds to acceptance (resp. rejection). Other criteria for acceptance could be considered as seen from the proofs of [6]. The fact that the acceptance region is at some distance from the rejectance region (a value between -1 and 1 means the absence of decision) is here only to avoid representation problems if one wants to simulate the involved equations.

Notice that [6] was leaving open the issue whether the involved polynomial in the polynomial ordinary differential equations could have non-rational coefficients (notice that the constructions were however using only computable coefficients, but possibly irrational). It has been proved recently that only rational coefficients are needed [3].

The notion of uniform computability is the strong notion of Turing universality involved in the rest of this paper.

3 Turing Completeness of Elementary Chemical Reaction Networks

The previous results provide a solid foundation for studying biochemical analog computation. However, a biochemical reaction system is a *positive* dynamical

system living in the cone $\mathbb{R}^n_+$, where the state is defined by the positive concentration values of the molecular species[9]. Furthermore, we wish to restrict ourselves to elementary reaction systems, governed by the mass-action-law kinetics and where each reaction has at most two reactants.

Let $\mathcal{M}$ be a *finite set* of n molecular species $\{y_1, \ldots, y_n\}$.

Definition 6. *[21] A* reaction *is a triple (R, P, f), where $R : \mathcal{M} \to \mathbb{N}$ is a multiset of reactants, $P : \mathcal{M} \to \mathbb{N}$ is a multiset of products and $f : \mathbb{R}^n_+ \to \mathbb{R}_+$, called the* rate function*, is a partially differentiable function verifying $R(y_i) > 0$ iff $\frac{\partial f}{\partial y_i}(y) > 0$ for some $y \in \mathbb{R}^n_+$.*

A reaction system *is a finite set of reactions.*

A mass-action-law reaction *is a reaction in which the rate function f is a monomial of the form $k * \Pi_{y \in \mathcal{M}} y^{R(y)}$ where k is called the* rate constant*.*

An elementary reaction *is a mass-action-law reaction with at most two reactants.*

For the sake of both readibility and reproducibility, the examples will be noted in the sequel in Biocham syntax, where a reaction (R, P, f) is written `f for R => P`, or just `R => P` if the rate function is a mass action kinetics with rate constant is equal to 1; the multisets are written with linear expressions and `_` stands for the empty multiset. Furthermore, a reaction with catalysts `f for R+C => C+P` is abbreviated as `f for R =[C]=> P`.

Definition 7. *The differential semantics of a reaction system $\{(R_i, P_i, f_i)\}_{i \in I}$ is the ODE system*

$$\{y' = \Sigma_{i \in I}(R_i(y) - Pi(y)) * f_i\}_{y \in \mathcal{M}}.$$

The dynamics given by the law of mass action leads to a polynomial ODE system of the form $y'(t) = p(y(t))$ with $p(y)_i = \sum_j (P_j(y_i) - R_j(y_i) * k_j * \Pi^n_{i=1} {y_i}^{R_j(y_i)}$. There are thus additional constraints, compared to general PIVPs: the components y_i must always be positive, and the monomials of p_i whose coefficient is negative must have a non-zero y_i exponent. These constraints are necessary conditions for the existence of a set of biochemically realizable reactions that react according to the dynamics $y' = p(y)$. Note however that we shall not discuss here the choice of their possible implementations by particular biochemical devices, such as DNA polymers [40], DNA double strands [33] or enzymatic reactions [19,37] as this is beyond the scope of this paper.

Interestingly, the previous computability and complexity results can be generalized to elementary biochemical reaction systems. First, the restriction to positive systems can be shown complete, by encoding each component y_i by the difference between two positive components y_i^+ and y_i^-, which can be normalized by a mutual annihilation reaction, $y_i^+ + y_i^- \Rightarrow \_$, so that one variable is null. It is worth noting that this encoding has been used in [36] for implementing linear I/O systems.

[9] Note that we do not impose that concentration values are small values, less than 1

Definition 8. *A function $f : \mathbb{R}_+ \to \mathbb{R}_+$ is* chemically-computable *if there exist a mass-action-law reaction system $\{(R_i, P_i, f_i)\}_{i\in I}$ over some molecular species $\{y_1, ..., y_n\}$, and a polynomial $q \in {\mathbb{R}_+}^n[\mathbb{R}_+]$ defining the initial concentration values, such that f is GPAC-computed by q and its (polynomial) differential semantics $p \in {\mathbb{R}_+}^n[{\mathbb{R}_+}^n]$.*

A function $f : \mathbb{R}_+ \to \mathbb{R}$ is chemically-computable *if there exists a chemically computable function $f^+ : \mathbb{R}_+ \to {\mathbb{R}_+}^2$ (by straightfortward generalization of Def.4 to multiple computations) over $\{y_1^+, ..., y_n^+, y_1^-, ..., y_n^-\}$ such that $f = f_1^+ - f_2^-$.*

In this definition, to compute $f(x)$, one has thus to design a reaction system over a finite set of molecular species, initialized to some values defined by a vector of polynomials $q(x)$ (e.g. following [8,12]), which guarantees that the result is obtained in the concentration of one distinguished molecular species, with a precision indicated by another distinguished molecular species (see Def. 4). Note however that in practice, in the examples of the following sections, the precision parameter will be left.

How to design such a reaction system is shown by the proofs of the following results.

Theorem 3. *Any GPAC-computable function can be computed by a mass-action-law reaction system under the differential semantics preserving the polynomial length complexity.*

Proof. Let us consider a GPAC-computable function by a polynomial differential equation $p \in \mathbb{R}^n[\mathbb{R}^n]$. Each variable $y_i \in \mathbb{R}$ can be encoded by a couple of variables $(y_i^+, y_i^-) \in \mathbb{R}_+^2$ such that at any time, $y_i = y_i^+ - y_i^-$.

Let $\hat{p}_i(y_1^+, y_1^-, \ldots, y_n^+, y_n^-) = p_i[y = y^+ - y^-]$, we write $\hat{p}_i = \hat{p}_i^+ - \hat{p}_i^-$, where the monomials of $\hat{p}_i^+$ and $\hat{p}_i^-$ have positive coefficients. A positive system is then defined by:

$$\forall i \leq n,\ \begin{cases} {y_i^+}' &= \hat{p}_i^+ - f_i y_i^+ y_i^- \\ {y_i^-}' &= \hat{p}_i^- - f_i y_i^+ y_i^- \\ y_i^+(0) &= \max(0,\ y_i(0)) \\ y_i^-(0) &= \max(0, -y_i(0)) \end{cases}$$

where the f_i's are polynomials with positive coefficients such that $f_i \geq \max(\hat{p}_i^+, \hat{p}_i^-)$, for instance $f_i = \hat{p}_i^+ + \hat{p}_i^-$. The terms $-f_i y_i^+ y_i^-$ can be implemented by annihilation reactions

$$f_i \ \ for\ \ y_i^+ + y_i^- \xrightarrow{y^+, y^-} .$$

which ensure that one of the $y_i^\pm$ always remains small.

Note that we have: ${y_i^+}' \leq \hat{p}_i^+(1 - y_i^+ y_i^-)$ and ${y_i^-}' \leq \hat{p}_i^-(1 - y_i^+ y_i^-)$, so that $(y_i^+ y_i^-)' \leq q \cdot (1 - y_i^+ y_i^-)$ where q is a polynomial with positive coefficients. Since at $t = 0$ we have $y_i^+ y_i^- = 0$, we deduce by a Gronwall inequality that we always have $y_i^+ y_i^- \leq 1$. Therefore, $|y_i^\pm| \leq |y_i| + 1$, and $|y^\pm| \leq |y| + n$. Consequently, if the original system is increased in space by a polynomial in the

for instance. We consider arbitrary large concentration and molecule numbers [25].

size of the input and the time, then this is still the case for the positive system obtained by the preceding construction. Furthermore, each monoid of the form $\lambda y_1^{\alpha_1} \dots y_m^{\alpha_m}, \lambda > 0$ appearing in the right term of an equality of the form $y = p$ can be implemented by a reaction of the form

$$\alpha_1 y_1 + \ldots + \alpha_m y_m \xrightarrow{\lambda} y + \alpha_1 y_1 + \ldots + \alpha_m y_m.$$

□

Second, one can remark that we can also restrict ourselves to elementary reactions, since every PIVP is equivalent to a quadratic PIVP:

Theorem 4. *[11] Any solution of a PIVP is the solution of a PIVP of degree at most two.*

Proof. The proof consists in introducing variables for each monomial as follows

$$v_{i_1,\ldots,i_n} = y_1^{i_1} y_2^{i_2}, \ldots, y_n^{i_n}.$$

We have $y_1 = v_{1,0,\ldots,0}$ and so on. The substitution of these variables in the differential equations of y_i' gives equations of the first degree in the variables $v_{i_1,\ldots,i_n}$. The differential equations for variables that are not y_i are of the form

$$v'_{i_1,\ldots,i_n} = \sum_{k=0}^{n} i_k * v_{i_1,\ldots,i_k-1,\ldots,i_n} * y'_k$$

i.e. a polynomial of degree two since the y'_k differentials are linear combinations of the variables $v_{i_1,\ldots,i_n}$. □

These results show that elementary biochemical reaction systems under the differential semantics have the expressive power of PIVPs. By Theorems 1, 3 and 4, we get

Theorem 5. *Elementary reaction systems on finite universes of molecules are Turing-complete under the differential semantics.*

It is worth noticing that this result differs from previous results on the universality of continuous chemical reaction networks or neural networks which were based on a *non-uniform* notion of computability [34,27]. Here we obtain a *uniform computability* result: a given reaction system on a finite set of molecular species is able to simulate a Turing machine on all inputs, independently of the size of the input. This result can be considered as solving the open problem mentionned explicitly in Section 8 of [17].

Furthermore, our translation of PIVPs to positive quadratic PIVPs preserves the polynomial time complexity defined in PIVPs as the trajectory length up to some precision. The translation of Theorem 2 together with Theorem 3 and 4 give

Theorem 6. *A function over the reals is computable (resp. in polynomial time) if and only if it is computable by an elementary reaction system using only synthesis reactions with at most two catalysts of the form*

```
- => z
```
or
```
_ =[x]=> z
```
or
```
_ =[x+y]=> z
```

and degradation reactions by annihilation of the form

```
x_p + x_m => _
```

(resp. with trajectories of polynomial length).

Proof. In the proof of Theorem 3, we have shown that one consequence of the annihilation reactions with fast kinetics is to make x_p and x_m not larger than $|x|+1$ for all x, and thereby ensure the preservation of the polynomial complexity. This inequality also shows that annihiliation reactions are useful to ensure the convergence of the result components.

One can remark in this proof that the encoding of real valued variables by two signed variables allows us to replace substractions by additions in the ODEs just by sorting the monomials according to their sign. Furthermore, the proof of Theorem 4 rewrites the terms with terms of degree at most 2 without changing their sign. As a consequence, all the terms of the ODE are monomials of the forms k, $k * x$, $k * x * y$ or $-f * x_p * x_m$ which can be encoded with synthesis reactions with at most two catalysts, and annihiliation reactions. □

The possible implementations of the particular synthesis and degradation reactions used in Theorem 6 are beyond the scope of this paper. Let us just remark that a formal synthesis reaction as `_ =[x]=> z` does not need to be a real synthesis reaction with DNA or RNA, but can be implemented with proteins, for instance by a phosphorylation reaction by kinase x, i.e. of the form `iz =[x]=> z` where iz assumed to be in excess is the (inactive) dephosphorylated form of z. Similarly, the annihilation reaction `z_p + z_m => _` might be thought as representing in reality, among many other possibilities, a complexation reaction which produces an inactive (stable) complex.

4 Biochemical Compilation of Analog Functions

4.1 Compilation of GPAC-Generable Functions

The proof of Theorem 3 shows how a PIVP can be implemented with biochemical reactions by doubling the number of variables for the positive and negative parts, and by implementing each monomial of the differential equations by a catalytic reaction of synthesis or degradation according to its sign. Similarly, the proof of Theorem 4 shows how to restrict code generation to elementary reactions of at most two reactants, by increasing the number of variables (i.e. molecular species), that is to say by sacrificing the dimension of the system to the minimization of the degrees.

These are the principles of our biochemical compiler which translates a mathematical function defined by a PIVP into a system of elementary reactions. For implementation reasons however, our compiler departs from the previous theoretical framework in a few places. The annihilation reactions (which play no role in the computability but in the complexity only) are implemented with a sufficiently large rate constant called *fast*, instead of with a large polynomial. The approximation error is not computed since we are not interested in the precision of the result and assume to know in advance some time horizon sufficient to get the results[10].

As a first example, let us consider the biochemical compilation of the oscillator defined by the cosine function $f = cos(t)$ as a function of time, itself defined by the PIVP $f'' = -f$ with $f(0) = 1$, i.e. $\{f' = z,\ z' = -f\}$ with $f(0) = 1$, $z(0) = 0$. This example compiles into the six elementary synthesis reactions below, where the first four reactions implement the PIVP, and the last two reactions the normalization reactions by mutual annihilation of the positive and negative variables.

```
biocham: compile_from_expression(cos, time, f).
  _ = [z2_p] => f_p.
  _ = [z2_m] => f_m.
  _ = [f_m] => z2_p.
  _ = [f_p] => z2_m.
  fast*z2_m*z2_p for z2_m+z2_p => _.
  fast*f_m*f_p for f_m+f_p => _.
  present (f_p, 1).
biocham: list_ode.
  d(f_p)/dt = z2_p-fast*f_m*f_p
  d(f_m)/dt = z2_m-fast*f_m*f_p
  d(z2_p)/dt = f_m-fast*z2_m*z2_p
  d(z2_m)/dt = f_p-fast*z2_m*z2_p
```

This reaction system, produced with initial concentration value $f_p = 1$ at time 0 (and 0 for all other variables), is designed for the differential semantics. Its robustness to extrinsic noise can be measured with respect to perturbations of the parameter values [42]. Such a reaction system can also be interpreted in the stochastic semantics [22], and simulated using Gillespie's SSA algorithm [24] to analyze its robustness to intrinsic noise. Figure 2 shows a differential simulation trace and one stochastic simulation trace.

4.2 Compilation of GPAC-Computable Functions

Let us first remak that a PIVP that *computes* the value of $y = f(x)$ at any point x can be derived from a PIVP that *generates* $f(t)$ as a function of time [39]. The idea is to replace the PIVP that generates $f(t)$ by a PIVP that generates $f(\gamma(t))$

[10] Note also that the transformation to at most binary reactions is temporarily not included in our compiler.

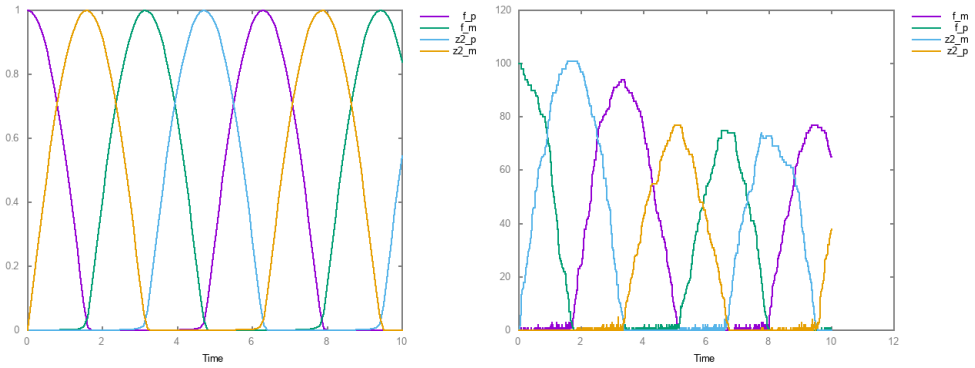

Fig. 2. Differential and stochastic simulation traces of the compiled reactions for generating the cosine function as a function of time.

where $\lim_{t\to\infty} \gamma(t) = x$, starting from a point x_0 such that $f(x)$ does not diverge along the trajectory $\gamma(t)$ [39]. Taking the trajectory $\gamma(t) = x + (x_0 - x)e^{-\lambda t}$ with $\lambda > 0$, we have $\gamma(t)' = -(x_0 - x)e^{-\lambda t} = x - \gamma(t)$.

Although not totally general since all GPAC-computable functions are not GPAC-generable, we limit ourselves to this method for compiling computable functions with the following

Algorithm 1 Transformation of a PIVP that generates a function $f(t)$ in a PIVP that computes the function $f(x)$ for any x as $f(\gamma(t))$.

1. replace t by $\gamma(t)$ in the ODE that generates the function $f(t)$;
2. multiply all the terms of the ODE by $x - \gamma(t)$;
3. add the equation $\gamma' = x - \gamma$;
4. initialize γ to x_0 and the result variable to $f(x_0)$.

For instance, the compilation of the cosine function $cos(x)$ for any input concentration x generates the following elementary synthesis reaction system, where the first four reactions compute $\gamma(t)$ in `g_p` and `g_m` (with $\lambda = 1$), and the other reactions result from the multiplication by $x - \gamma$ of the ODE terms for $cos(t)$ which basically translates to the addition of catalysts `x_p` and `g_m` to the reactions for $cos(t)$:

```
biocham: compile_from_expression(cos, x, r).
  _ = [g_m] => g_p.
  _ = [x_p] => g_p.
  _ = [g_p] => g_m.
  _ = [x_m] => g_m.
```

```
  _ = [g_m+z4_p] => r_p.
  _ = [g_p+z4_m] => r_p.
  _ = [x_m+z4_m] => r_p.
  _ = [x_p+z4_p] => r_p.
  _ = [g_m+z4_m] => r_m.
  _ = [g_p+z4_p] => r_m.
  _ = [x_p+z4_m] => r_m.
  _ = [x_m+z4_p] => r_m.
  _ = [g_m+r_m] => z4_p.
  _ = [g_p+r_p] => z4_p.
  _ = [x_p+r_m] => z4_p.
  _ = [x_m+r_p] => z4_p.
  _ = [g_m+r_p] => z4_m.
  _ = [g_p+r_m] => z4_m.
  _ = [x_m+r_m] => z4_m.
  _ = [x_p+r_p] => z4_m.
  fast*z4_m*z4_p for z4_m+z4_p => _.
  fast*r_m*r_p for r_m+r_p => _.
  fast*g_m*g_p for g_m+g_p => _.
  fast*x_m*x_p for x_m+x_p => _.
  present (r_p, 1).
biocham: present (x_p, 4).
```

This reaction system then computes $cos(x)$ by initializing the argument to the desired value, for instance $x_p = 4$ for which simulation traces are shown in Figure 3.

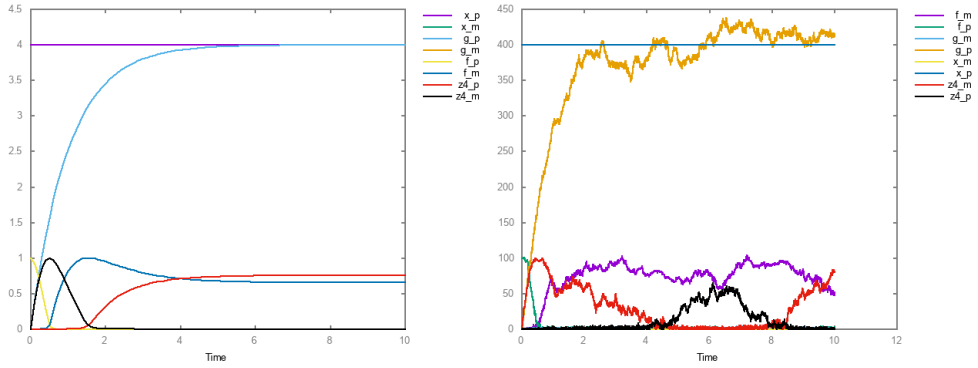

Fig. 3. Differential and stochastic simulation traces of the compiler-generated reactions for computing $cos(4)$.

5 Compilation of Sigmoid Functions

A sigmoid function is a bounded differentiable real function that is defined for all real input values and has a positive derivative at each point. Sigmoid functions have an "S" shape. They can be used to implement analog/digital converters which produce all-or-nothing outputs for a wide range of input levels. In biochemistry, Hill functions, of the form $x^n/(k+x^n)$, over $\mathbb{R}_+$ are examples of sigmoid functions that have been shown to approximate the input/output response of, first historically, cooperative allosteric enzymatic reactions [44], and more recently of the MAPK signaling network [28] for instance. In this section we study the biochemical compilation of various sigmoid functions which is key to the implementation of digital logic with molecular reactions [32,31].

5.1 Logistic, Hyperbolic Tangent, Arc Tangent and Hill Sigmoids

For the sake of simplicity, we restrict here to the generation of sigmoid functions as functions of time, with the idea of using Alg. 1 for computing those functions as functions of some input variable. The logistic function $S(t) = 1/(1+e^t)$ is a sigmoid function over $\mathbb{R}$ whose derivative can be written in terms of itself as $S'(t) = S(t) - S(t)^2$. It can be generated over $\mathbb{R}^+$ by two simple elementary reactions, one autocatalyzed synthesis and one autocatalyzed degradation:

```
S => 2*S.     S =[S]=> _.     present(S,0.5).
```

The hyperbolic tangent $tanh(t)$ has also a simple derivative expression $tanh'(t) = 1 - tanh(t)^2$ which can be implemented with two elementary reactions:

```
_ => HT.    2*HT => HT.
```

The arc tangent $atan(t)$ has for derivative $atan'(t) = 1/(1+t^2)$ which can be implemented by

```
_ => T.        1/(1+T^2) for /T => AT.
```

Note however that in this presentation, the second synthesis reaction uses `T` as reaction inhibitor, which is beyond the scope of this paper.

The Hill functions of degree d (resp. negative Hill functions) are defined by $H_d(t) = t^d/(k+t^d)$ (resp. $NH_d(t) = 1/(k+t^d)$) for some parameter $k \in \mathbb{R}$. One can easily check that they are solutions of the PIVP $H'_d = d*k*t^{d-1}*NH_d^2$, $NH'_d = -d*t^{d-1}*NH_d^2$ with $H_d(0) = 0$ and $NH_d(0) = 1/k$, which leads to the following (non elementary) reactions for their generation:

```
MA(d) for NHd =[(d-1)*T+NHd]=> _.    present(NHd,1/k).
MA(d*k) for _ =[(d-1)*T+2*NHd]=> Hd.
```

5.2 Comparison to MAPK Signaling Circuits

MAPK (mitogen-activated protein kinases) signaling networks are very common biochemical reaction modules which are found in multiple copies in eukaryotic organisms. In these signaling cascades the proteins activated by phosphorylation are themselves kinases which catalyze in cascade other phosphorylations. Thus, the MAPK cascade has three stages of phosphorylation for a total of 30 elementary reactions: the entry E_1 of the cascade, directly linked to the membrane

receptor, catalyses the phosphorylation of the kinase KKK of the first stage, which in turn phosphorylates the kinase KK of the second stage, which in this doubly phosphorylated form phosphorylates the protein K of the last stage of the cascade, which, when doubly phosphorylated in Kpp, is able to migrate into the nucleus and promote or inhibit gene transcription.

In [28] Huang and Ferrell have proposed an explanation for this structure by showing that the MAPK cascades exhibit a (stationary) response in the form of a Hill function which produces a nearly all-or-nothing response. That is, by denoting (u, y) the input-output relation of the system, they could approximate the dose-response diagram by an equation of the form $Y(u) \approx \lambda \frac{u^d}{c^d+u^d}$ with d in the order of 4.9 at the third level Kpp $d \sim 1.7$ to the second $KKpp$ and $d = 1$ at the first level $KKKp$.

The Hill function, as a function of an input, can be compiled in biochemical reactions by applying Alg. 1 to the PIVP given in the previous section for the Hill function as a function of time. This leads to the following reaction system:

$$\left\{\begin{array}{llll} \gamma & \to & \_ & y_2 + d + x + y_1 \to d + x + y_1 + 2y_2 \\ x & \to & x + \gamma & 2y_2 + d + x + y_1 \to d + x + y_1 + y_2 \\ 2y_1 + x & \to & y_1 + x & y_2 + d + \gamma + y_1 \to d + \gamma + y_1 \\ 2y_1 + \gamma & \to & 3y_1 + \gamma & 2y_2 + d + \gamma + y_1 \to d + \gamma + y_1 + 3y_2 \end{array}\right\}$$

with the initial conditions $(\gamma, y_1, y_2)_{t=0} = (1, 1, 1/2)$. This system satisfies $y_2 = \frac{x^d}{1+x^d}$ at steady state, and therefore constitutes a binary presence indicator: if $x \gg 1$, then $y_2 = 1$, and if $x \ll 1$, then $y_2 = 0$, the greater d, the greater the discrimination. Note that this value is given here by a fixed concentration of molecule but could be represented more simply by a kinetic constant. This converter, however, fails to create an intermediate value in $\frac{1}{\gamma}$ which gives an exponential amplitude for $x = 0$, and therefore an exponential computational complexity in the sense of the previous section. If we restrict ourselves to taking x in an interval of the form $[\varepsilon, +\infty[$, with $\varepsilon > 0$, then the complexity becomes polynomial. On the other hand, if we restrict to degree 2 and compile the expression $x^2/(1+x^2)$, the command `compile_from_expression(id*id/(1+id*id),x,y)` produces a system of 259 reactions over 23 species (70 reactions over 19 species for the function of time). However, the generated species for the possibly negative values, and their reactions, are useless in this example. Furthermore, our syntax-directed compilation strategy currently associates one variable per term occurrence, thus twice for the two occurrences of the expression x^2, and performs division in another variable. The computational complexity is polynomial, but with one component of amplitude x^2 which is computed in that strategy.

The natural MAPK circuit of 30 reactions [28] thus currently appears both more concise, and with a lesser computational complexity, than the system of reactions produced according to our first principles of compilation without any optimization.

6 Compilation of Sequentiality and Program Control Flows

The negative Hill sigmoid $\frac{c}{c+x^d}$ provides a binary absence indicator of higher quality than those proposed in in [45] or even [29] for implementing sequentiality and program control flows, for which leakage phenomena may occur: even in the relative absence of the x species, the presence indicator remains at a sufficiently high concentration to catalyze certain reactions, or the opposite effect, the absence indicator may be too small. This is particularly visible in the sequentiality implementation: given the R_i reactions, if we want R_2 to be executed only once R_1 is completed, one can impose an indicator of the absence of one species consumed by R_1 as catalyst of R_2, ditto between R_2 and R_3, *etc* This leads however to the following phenomenon: the reactions are made all the more slowly as i is large, in other words, the reactions accumulate delay in their execution due to the retention of absence indicators.

With a sufficiently powerful absence indicator, it is possible to implement the sequentiality, the conditional instruction, and loop structures of algorithmic programming. It has been shown in [29] how to compile small imperative programs into a system of biochemical reactions wherein the molecular species are used as markers of the position of the program in a control flow graph. This was illustrated with the compilation of Euclidean division and greatest common divisor programs, and with strategies for species minimization in [30].

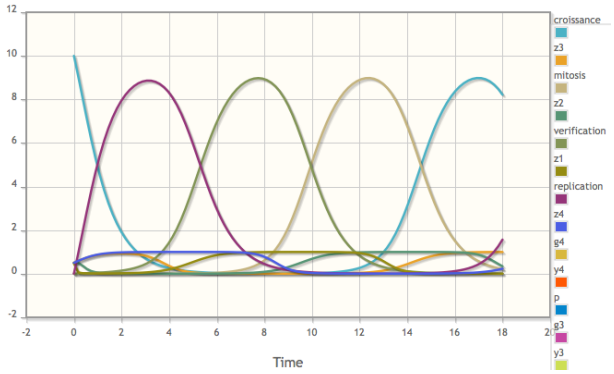

Fig. 4. ODE simulation trace of the generated reactions for the cell cycle loop.

Along the same lines, a minimalist specification of the cell division cycle can be specified by the program

```
while true do {Growth; Replication; Verification; Mitosis;}
```

The compilation of this program in elementary reactions implements the sequentiality of the four phases of the cycle by the degradation of the markers of each of the phases, depicted in Figure 4. Interestingly, the resulting simulation curves

are quite similar to the concentration curves obtained in cell cycle models [23] for the cyclin proteins, which appear here as necessary markers for implementing sequentiality with biochemical reactions.

7 Discussion and Perspectives

Though one lesson of Computer Science is that analog computation does not scale up, while digital computation does, the biological perspective provides a new impetus to the study of analog computation and mixed analog/digital parallel programs.

We have shown that recent results in computable analysis and theoretical complexity establish solid links between analog and digital computation, and can be used to compile analog specifications and mixed analog/digital programs into elementary biochemical reactions. This opens new research avenues to analyze natural protein interaction circuits not only from point of view of the size and the static complexity of the networks [1], but also from the computational complexity and robustness points of view [39], to revisit the important particular case of linear time invariant systems [16,15,14], to design reaction code optimizers, and compare natural circuits acquired by evolution to engineered and compiler-generated synthesized circuits.

The concept of biochemical computation and compilation can also be experimented *in vitro* and *in vivo*, either in Synthetic Biology, through the modification and reprogramming of living cells [35,18], or in Synthetic Biochemistry, through the creation and programming of non-living microfluidic vesicles [19], with various applications including the design of biomarkers [18].

Furthermore, the formal specification by mathematical functions of the input/output or transient behaviors of biochemical reaction systems under the differential semantics, establishes novel ways to study the functions of natural circuits mathematically, and on this route investigate their evolution history and evolution capabilities [47].

Acknowledgements. We are grateful to especially one reviewer for his expert proofreading which helped us to improve the presentation of our results, and to the editors for providing us with the necessary extra space. Part of this research is funded by the ANR-MOST Biopsy project. The first author acknowledges fruitful discussions with Jie-Hong Jiang (NTU, Taiwan) on the compilation of program control flows with reactions, and motivating discussions with Frank Molina (CNRS, Sys2Diag, Montpellier) on the biochemical implementation by enzymatic reactions in microfluidic vesicles.

References

1. Barabási, A.L.: Network Science. Cambridge University Press (2016)
2. Berry, G., Boudol, G.: The chemical abstract machine. Theoretical Computer Science 96 (1992)

3. Bournez, O., Graça, D.S., Pouly, A.: Polynomial Time corresponds to Solutions of Polynomial Ordinary Differential Equations of Polynomial Length. Journal of the ACM (2017), accepted for publication
4. Bournez, O., Campagnolo, M.L., Graça, D.S., Hainry, E.: Polynomial differential equations compute all real computable functions on computable compact intervals. Journal of Complexity 23(3), 317–335 (2007), https://hal-polytechnique.archives-ouvertes.fr/inria-00102947
5. Bournez, O., Campagnolo, M.L., Graça, D.S., Hainry, E.: The general purpose analog computer and computable analysis are two equivalent paradigms of analog computation. In: International Conference on Theory and Applications of Models of Computation. pp. 631–643. Springer (2006)
6. Bournez, O., Graça, D.S., Pouly, A.: Polynomial Time corresponds to Solutions of Polynomial Ordinary Differential Equations of Polynomial Length. The General Purpose Analog Computer and Computable Analysis are two efficiently equivalent models of computations. In: 43rd International Colloquium on Automata, Languages, and Programming, ICALP 2016, July 11-15, 2016, Rome, Italy. LIPIcs, vol. 55, pp. 109:1–109:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016), http://drops.dagstuhl.de/opus/frontdoor.php?source_opus=6244
7. Bournez, O., Graça, D.S., Pouly, A.: On the functions generated by the general purpose analog computer. Information and Computation (accepted under minor revision) (2017)
8. Buisman, H.J., ten Eikelder, H.M.M., Hilbers, P.A.J., Liekens, A.M.L.: Computing algebraic functions with biochemical reaction networks. Artificial Life 15(1), 5–19 (2009)
9. Busi, N., Gorrieri, R.: On the computational power of brane calculi. In: Plotkin, G. (ed.) Transactions on Computational Systems Biology VI, Lecture Notes in BioInformatics, vol. 4220, pp. 16–43. Springer-Verlag (Nov 2006), cMSB'05 Special Issue
10. Cardelli, L., Zavattaro, L.: Turing universality of the biochemical ground form. Mathematical Structures in Computer Science 20(1), 45–73 (2010)
11. Carothers, D.C., Parker, G.E., Sochacki, J.S., Warne, P.G.: Some properties of solutions to polynomial systems of differential equations. Electronic Journal of Differential Equations 40 (2005)
12. Chen, H.L., Doty, D., Soloveichik, D.: Rate-independent computation in continuous chemical reaction networks. In: Proceedings of the 5th Conference on Innovations in Theoretical Computer Science. pp. 313–326. ITCS '14, ACM, New York, NY, USA (2014)
13. Chen, Y., Dalchau, N., Srinivas, N., Phillips, A., Cardelli, L., Soloveichik, D., Seelig, G.: Programmable chemical controllers made from dna. Nature Nanotechnology 8, 755–762 (Sep 2013)
14. Chiang, H.J., Jiang, J.H., Fages, F.: Reconfigurable neuromorphic computation in biochemical systems. In: Proc. 37th Annual International Conference of the IEEE Engineering in Medicine and Biology Society EMBC (2015), http://lifeware.inria.fr/~fages/Papers/CJF15ieee.pdf
15. Chiang, K., Jiang, J.H., Fages, F.: Building reconfigurable circuitry in a biochemical world. In: BioCAS'14: IEEE Biomedical Circuits and Systems Conference. IEEE, Lausanne, Switzerland (Oct 2014), http://lifeware.inria.fr/~fages/Papers/CJF14biocas.pdf
16. Chiu, T.Y., Chiang, H.J.K., Huang, R.Y., Jiang, J.H.R., Fages, F.: Synthesizing configurable biochemical implementation of linear systems from their transfer function specifications. PLoS ONE 10(9) (2015)

17. Cook, M., Soloveichik, D., Winfree, E., Bruck, J.: Programmability of chemical reaction networks. In: Condon, A., Harel, D., Kok, J.N., Salomaa, A., Winfree, E. (eds.) Algorithmic Bioprocesses, pp. 543–584. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
18. Courbet, A., Endy, D., Renard, E., Molina, F., Bonnet, J.: Detection of pathological biomarkers in human clinical samples via amplifying genetic switches and logic gates. Science Translational Medicine (2015)
19. Courbet, A., Amar, P., Fages, F., Renard, E., Molina, F.: Computer-aided biochemical programming of synthetic microreactors operating as logic-gated and multiplexed diagnostic devices. Submitted
20. Daniel, R., Rubens, J.R., Sarpeshkar, R., Lu, T.K.: Synthetic analog computation in living cells. Nature 497(7451), 619–623 (05 2013)
21. Fages, F., Gay, S., Soliman, S.: Inferring reaction systems from ordinary differential equations. Theoretical Computer Science 599, 64–78 (Sep 2015), `http://lifeware.inria.fr/~fages/Papers/FGS14tcs.pdf`
22. Fages, F., Soliman, S.: Abstract interpretation and types for systems biology. Theoretical Computer Science 403(1), 52–70 (2008), `http://lifeware.inria.fr/~fages/Papers/FS07tcs.pdf`
23. Gérard, C., Goldbeter, A.: Temporal self-organization of the cyclin/cdk network driving the mammalian cell cycle. Proceedings of the National Academy of Sciences 106(51), 21643–21648 (Dec 2009)
24. Gillespie, D.T.: General method for numerically simulating stochastic time evolution of coupled chemical-reactions. Journal of Computational Physics 22, 403–434 (1976)
25. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Journal of Physical Chemistry 81(25), 2340–2361 (1977)
26. Graça, D., Costa, J.: Analog computers and recursive functions over the reals. Journal of Complexity 19(5), 644–664 (2003)
27. Helmfelt, A., Weinberger, E.D., Ross, J.: Chemical implementation of neural networks and turing machines. PNAS 88, 10983–10987 (1991)
28. Huang, C.Y., Ferrell, J.E.: Ultrasensitivity in the mitogen-activated protein kinase cascade. PNAS 93(19), 10078–10083 (Sep 1996)
29. Huang, D.A., Jiang, J.H., Huang, R.Y., Cheng, C.Y.: Compiling program control flows into biochemical reactions. In: ICCAD'12: IEEE/ACM International Conference on Computer-Aided Design. pp. 361–368. ACM, San Jose, USA (Nov 2012), `http://lifeware.inria.fr/~fages/Papers/iccad12.pdf`
30. Huang, R.Y., Huang, D.A., Chiang, H.J.K., Jiang, J.H., Fages, F.: Species minimization in computation with biochemical reactions. In: IWBDA'13: Proceedings of the fifth International Workshop on Bio-Design Automation. Imperial College, London (Jul 2013), `http://lifeware.inria.fr/~fages/Papers/HHCJF13iwbda.pdf`
31. Jiang, H., Riedel, M., Parhi, K.K.: Digital signal processing with molecular reactions. IEEE Design and Test of Computers 29(3), 21–31 (Jun 2012)
32. Jiang, H., Riedel, M., Parhi, K.K.: Digital logic with molecular reactions. In: ICCAD'13: IEEE/ACM International Conference on Computer-Aided Design. pp. 721–727. ACM (Nov 2013)
33. Lakin, M.R., Parker, D., Cardelli, L., Kwiatkowska, M., Phillips, A.: Design and analysis of dna strand displacement devices using probabilistic model checking. Journal of the Royal Society Interface 9(72), 1470–1485 (2012)
34. Magnasco, M.O.: Chemical kinetics is turing universal. Physical Review Letters 78(6) (1997)

35. Nielsen, A.A.K., Der, B.S., Shin, J., Vaidyanathan, P., Paralanov, V., Strychalski, E.A., Ross, D., Densmore, D., Voigt, C.A.: Genetic circuit design automation. Science 352(6281) (2016)
36. Oishi, K., Klavins, E.: Biomolecular implementation of linear i/o systems. IET SYstems Biology 5(4), 252–260 (2011)
37. P.Arkin, A., Ross, J.: Computational functions in biochemical reaction networks. Biophysical Journal 67, 560–578 (1994)
38. Paun, G., Rozenberg, G.: A guide to membrane computing. Theoretical Computer Science 287(1), 73–100 (2002)
39. Pouly, A.: Continuous models of computation: from computability to complexity. Ph.D. thesis, Ecole Polytechnique (Jul 2015)
40. Qian, L., Soloveichik, D., Winfree, E.: Efficient turing-universal computation with dna polymers. In: Proc. DNA Computing and Molecular Programming. LNCS, vol. 6518, pp. 123–140. Springer-Verlag (2011)
41. Rizik, L., Ram, Y., Danial, R.: Noise tolerance analysis for reliable analog and digital computation in living cells. J Bioengineer & Biomedical Sci 6(186) (2016)
42. Rizk, A., Batt, G., Fages, F., Soliman, S.: Continuous valuations of temporal logic specifications with applications to parameter optimization and robustness measures. Theoretical Computer Science 412(26), 2827–2839 (2011), `http://lifeware.inria.fr/~soliman/publi/RBFS11tcs.pdf`
43. Sauro, H.M., Kim, K.: Synthetic biology: It's an analog world. Nature 497(7451), 572–573 (05 2013)
44. Segel, L.A.: Modeling dynamic phenomena in molecular and cellular biology. Cambridge University Press (1984)
45. Senum, P., Riedel, M.: Rate-independent constructs for chemical computation. PLOS One 6(6) (2011)
46. Shannon, C.: Mathematical theory of the differential analyser. Journal of Mathematics and Physics 20, 337–354 (1941)
47. Valiant, L.: Probably Approximately Correct. Basic Books (2013)
48. Weihrauch, K.: Computable Analysis: an Introduction. Springer (2000)